Project Documentation

Team name: Berozgar Mandali

Project Overview

This project implements an algorithm visualization application using Qt and Box2D frameworks. The application, called AlgoCrash, allows users to visualize and interact with common sorting algorithms through a physics-based interface. Users can observe how different sorting algorithms work step-by-step, understand their behavior, and compare their performance metrics through an engaging visual representation.

Architecture

The project follows an object-oriented design with clear separation of concerns. The main components work together to create a physics-based sorting visualization system. The main components are:

Core Classes

1. **PhysicsBlock** - Represents a physical block with a numerical value
    - Combines Box2D physics with Qt graphics
    - Provides methods for positioning, movement, and highlighting
    - Manages synchronization between physics and graphical representations
2. **SortingController** - Manages the execution of sorting algorithms
    - Implements multiple sorting algorithms (Bubble, Insertion, Selection)
    - Tracks performance statistics (comparisons, swaps)
    - Provides step-by-step visualization control
    - Generates status updates for UI display
3. **MainWindow** - Main interface for the application
    - Manages the UI components and their interactions
    - Controls the physics simulation using Box2D
    - Handles user input for controlling the visualization
    - Provides controls for customizing the data to be sorted

Physics Integration

1. **Box2D World** - Physics simulation environment
    - Simulates gravity and physical interactions

    - Creates and manages rigid bodies for blocks
    - Applies forces and constraints to objects
2. **Physics Synchronization**
    - Maps physics coordinates to screen coordinates
    - Updates visual representation based on physics state
    - Controls physics parameters during sorting operations

UI Components

1. **Graphics Scene** - Visual representation area
    - Displays the PhysicsBlock objects
    - Provides viewing and scaling capabilities
2. **Control Panel** - User interface for algorithm control
    - Algorithm selection dropdown
    - Speed control slider
    - Step, Start/Pause, Reset, and Customize buttons
3. **Information Display**
    - Algorithm explanation panel
    - Sorting status updates
    - Performance statistics (comparisons and swaps)
    - Completion indicator

Features

- **Multiple Sorting Algorithms**: Bubble Sort, Insertion Sort, and Selection Sort with visual differentiation
- **Physics-Based Visualization**: Blocks with physics properties for realistic movement
- **Step-by-Step Execution**: Ability to advance algorithms one step at a time
- **Automatic Sorting**: Option to run the sorting process automatically at adjustable speeds
- **Customizable Data**: Input custom sets of integers for sorting
- **Performance Metrics**: Real-time tracking and display of comparison and swap counts
- **Algorithm Information**: Detailed explanations of each algorithm with complexity information
- **Visual Indicators**: Color highlighting to show active comparisons and sorted elements
- **Reset Functionality**: Ability to restart the visualization with the same or different data

File Structure

- **main.cpp**: Application entry point with Box2D test code

- **mainwindow.h/cpp**: Main UI window implementation and physics world management
- **physicsblock.h/cpp**: Implementation of the physical blocks used for visualization
- **sortingcontroller.h/cpp**: Implementation of sorting algorithms and visualization control

Sorting Algorithms

1. **Bubble Sort**
   - Compares adjacent elements and swaps if needed
   - Repeats until no more swaps are needed
   - Complexity: $O(n^2)$, Stable: Yes
   - Intuition: Heavy values bubble to the top
2. **Selection Sort**
   - Repeatedly selects the smallest element and places it in position
   - Divides the array into sorted and unsorted portions
   - Complexity: $O(n^2)$, Stable: No
   - Intuition: Picks the smallest and places it correctly
3. **Insertion Sort**
   - Builds the sorted array one item at a time
   - Inserts each element into its correct position
   - Complexity: $O(n^2)$, Stable: Yes
   - Intuition: Like sorting playing cards

Build and Deployment

The project uses the Qt build system with qmake. To build the project:

1. Ensure Qt and Box2D libraries are properly installed
2. Open the project in Qt Creator
3. Configure the project for your Qt version
4. Build and run using the buttons in Qt Creator

The application requires the following dependencies:

- Qt Framework (for UI and graphics)
- Box2D Physics Engine (for physics simulation)

Physics Simulation

The application uses Box2D for realistic physics simulation:

- Gravity affects blocks when they're not being sorted

- Blocks have physical properties like density, friction, and restitution
- Controlled animation during sorting operations
- Automatic synchronization between physics state and visual representation

Visualization Techniques

- **Color Coding**: White for unsorted, yellow for active comparison, green for sorted
- **Animated Movement**: Smooth transitions when swapping elements
- **Freeze Physics**: Gravity disabled during sorting for better visualization
- **Random Initial Placement**: Slight randomization in initial block positions for visual appeal

Future Improvements

Potential areas for enhancement:

1. Additional sorting algorithms (Quick Sort, Merge Sort, Heap Sort)
2. More detailed algorithm analysis and comparison
3. Sound effects to accompany visual operations
4. Adjustable visualization parameters (block size, gravity, etc.)
5. 3D visualization option with improved physics
6. Educational quiz mode to test algorithm understanding

Testing

The application has been manually tested for:

- Correct implementation of sorting algorithms
- Physics simulation stability
- UI responsiveness and control functionality
- Custom data input handling
- Performance with various data sets

Style Guidelines

All code follows the style guidelines defined in the project's C++ Code Style Guide. Key points include:

- Class names use PascalCase (e.g., MainWindow, PhysicsBlock)
- Method names use camelCase (e.g., getValue(), syncWithPhysics())
- Member variables use m_prefix (e.g., m_value, m_isComplete)
- Consistent indentation with 4 spaces
- Comprehensive documentation for classes and methods
- File headers with reviewer information
- Clear separation of implementation and declaration